

# matRiks: An R package for the automatic generation of rule-based matrices

by Ottavia M. Epifania and Bounciest Bilby

**Abstract** Few resources are available for the automatic generation of Raven-like matrices. Some of them are no longer working, while others are hardly customizable without advanced programming skills. Although an R package exists for generating stimuli for psychological assessments, it is currently confined to create rotation of the same shape. The [matRiks](#) package has been developed with the aim of overcoming the above mentioned issues. This package generates matrices according to different types of rules, from the most basic ones based on the visuo spatial features of the figures to the most complex ones, based on inferential and inductive reasoning. This unveils the possibility of generating new customizable stimuli and of systematically manipulating the complexity of the stimuli. Being developed within the R environment, the [matRiks](#) package is completely open-source, allows for the reproducibility of the stimuli, and it can be easily used by people with basic knowledge of the R language.

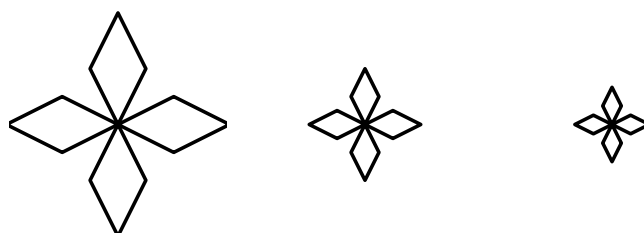
## 1 Introduction

Sievert (2020) cattell1963 defined fluid intelligence ( $g$ ) as the ability of solving novel reasoning problems that has little to do with concepts learned in schools or through acculturational processes. The adjective “fluid” explicitly refers to its ability to “flow” into a variety of tasks and cognitive activities (Horn, 1972). Given this definition of fluid intelligence, it appears natural that the instruments used for its evaluation tap on the respondent’s ability to solve abstract problems that involve acculturation as little as possible, such as figural analogies, figure classifications, matrices, and number and letter series (Horn, 1968).

The Raven’s progressive matrices (RPM, raven1938) are among the most famous tools for the assessment of  $g$ . The RPM consists in a series of non-verbal multiple-choice stimuli where respondents are required to complete a series of drawings composed of different figures by identifying the relevant features that rule the relationships between the figures. These drawings are often referred to as matrices. To pursue this aim, the respondents must choose the figure that complete the drawing among a list of other figures, the so-called distractors. This task should measure the ability of the respondents to identify and take into account the features (also called “rules”) that govern the relationship between the figures to compose the drawing. Different versions of the RPM exist, according to the target population (i.e., children with less than 12 years of age or adults) to which they are administered. The Colored Progressive Matrices (CPM, cit) are composed of sets of  $2 \times 2$  matrices (i.e., 4-cell matrices), some of which (CONTROLLARE) includes colored figures. The advanced progressive matrices (ADM, cit) are composed of sets of  $3 \times 3$  matrices meant for assessment in gifted population (DIRE MEGLIO). Finally, the RPM are meant for the assessment among the general population and are composed of both  $2 \times 2$  and  $3 \times 3$  matrices. The colored figures are present only in the CPM.

The RPM and similar tasks (here denoted as Raven-like matrices or Raven-like tasks) are employed in different fields, from clinical evaluation of intelligence to the selection processes in organizational psychology (citation needed). Since Raven’s and Raven-like tasks involve the ability to solve new abstract problems, the stimuli composing these tasks should not be spread among the general population to preserve the novelty of the tasks. The rules that govern the relationship between the figures of a matrix can also be used to generate new stimuli, and different resources have been developed throughout the years with this specific aim, such as Sandia (cit), Corvus (cit), and the R package [Imak](#) (cit).

The stimuli generated with Sandia have been analysed in an Item Response Theory framework to validate them as a test for measuring fluid intelligence. The stimuli are available upon request to the authors, however no new stimuli can be generated because the code on which Sandia is based is no longer maintained. Corvus represents another possible resource for generating Raven-like tasks. Corvus is written in Javascript but the Author provided a nice and easy-to-use graphical interface where the user can specify the figures and the rule(s) for the generation of the matrices. However, Corvus provides few degrees of freedom in terms of both the figures and the number of rules that can be manipulated through the graphical interface. Any customization from the user, like adding other figures, modifying existing ones, or implementing new rules, require to modify the JavaScript code, which might be quite demanding for people with little to no experience in JavaScript coding. Finally, the [Imak](#) package is an R package that allows for generating visual analogies. The code for generating such stimuli (along with their response options) is quite straightforward and easy to use. However,



**Figure 1:** Example of visuospatial rule: Changes in size

the stimuli that can be generated with the *ImaK* package are mostly based on the rotation of the same figure to which some objects can be added or removed. As such, the only rule that is manipulated is the spatial rotation of the figures.

Given the limitations of the existing resources for generating Raven-like tasks, there is the need of an open-source, easy-to-use, and constantly maintained resource for generating such stimuli through the systematic manipulation of rules applied to different figures. The *matRiks* package (*matRiks*) has been developed to pursue these aims. The package enables the generation of matrices by manipulating one or multiple rules on one or multiple figures. Additionally, it automatically generates the response list associated with the matrix. Beyond the default settings, the *matRiks* package allows for the generation of new figures and customization of matrices and response lists. The systematic manipulation of both the rules and the figures for the matrix generation should grant the possibility of grading the granularity of the complexity of the matrices by varying one element at the time. In a similar vein, the package should allow for generating matrices that can be considered equivalent in terms of rules employed for their generation but that differ in terms of figures composing the drawing. In what follows, the term stimulus is used to identify the matrix with its associated response list.

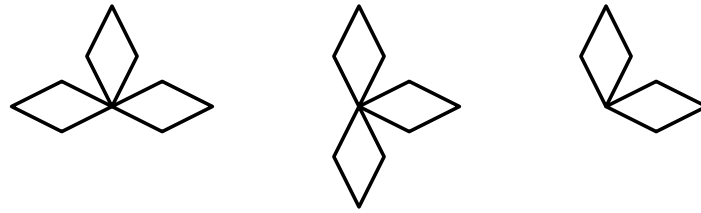
The manuscript is organized as follows. The next section presents the rules that are usually employed in the RPM along with the specific types of error responses (i.e., distractors) that compose the response list associated with a matrix. A complete example of the generation of a stimulus (i.e., the matrix and the associated response list) and some final remarks on the potential applications of this package conclude the presentation.

## 2 Background

### Rule based matrices

Literature highlights a plethora of rules that can be manipulated for the generation of the Raven-like tasks (cit cit cit). Beyond the fact that some of these rules have different names in different sources but refer to the same manipulation (e.g., the rule defined as “and problem” in Harris et al. 2020 is called “intersection” rule in Arendasy et al. 2005), they can be summarized into different macro-categories, namely visuospatial rules (i.e., the manipulation concerns the graphical and spatial features of the figures, Figure 1), and logical rules (i.e., the manipulation concerns the logical relationships between the figures composing the matrix, Figure 2). The rules can be used to manipulate to different figures to generate a matrix.

In Figure 1, the manipulation concerns a specific feature of the figure, that is its size, and it can be observed as the figure decreases its size across the cells. The leftmost cell contains the figure with its original size, the middle cell contains a smaller figure, while the rightmost cells contains the smallest one. In Figure 2, the manipulation concerns the relationships between the objects composing the figures, which are combined together according to a logical rule based on the insiemistic intersection of the objects. Specifically, the figure in the rightmost cell results from the intersection between the figures in the leftmost cell and those in the middle cell.



**Figure 2:** Example of logical rule: Insiemistic Interscetion AND

Both visuospatial and logical rules can be manipulated according to directional logic. Specifically, the rules can be applied horizontally (i.e., the manipulation of the rule can be seen across columns but not across rows, H direction), vertically (i.e., the manipulation of the rule can be seen across rows but not across columns, V direction), or diagonally (i.e., the manipulation of the rule can be seen both across columns and across rows). Concerning the diagonal directional logic, it can follow either the main diagonal of the matrix (i.e., the manipulation of the rule can be seen from the top-left corner to the low-right corner, TL-LR direction) or the secondary diagonal of the matrix (i.e., the manipulation of the rule can be seen from the low-left corner to the top-right corner, LL-TR direction).

### The response options

A large corpus of literature has investigated the role of the distractors in the response processes involved when solving the Raven matrices, focusing on the specific error responses chosen by the respondent (fort, kunda, storme). The underlying logic is that the incorrect response is not chosen at random by the respondent, but it can be the result of an educated guess, or it can be chosen because the respondent is misled by a specific feature. In other words, the incorrect responses might reflect an incorrect solution strategy resulting in the choice of a specific type of distractor over another one (Kunda et al. 2016). The distractors can be classified according to the incorrect response strategy they represent. Kunda et al. (2016) present a list of criteria for the identification of the distractors in the SPM based on classification of the error types from the CPM and APM manuals (Raven and Raven 2004). The distractor that is chosen in place of the correct response option can be collected into four main four conceptual errors: (i) Repetition (R) errors occur when the chosen response option is a cell adjacent to the blank space, (ii) Difference (D) errors occur when the chosen response option is completely different from any entry of the matrix, (iii) Wrong Principle (WP) errors occur when the chosen response option follows rules other than the ones used in the matrix, and (iv) Incomplete Correlate (IC) errors occur when the chosen response option is in fact the correct response with a variation on a single feature.

The characteristics of the specific error response can be categorized in various ways. For example, the repetition category can be divided based on the position of the repeated cell relative to the blank cell. Three error types within the R category can be identified: R-Left, R-Top, and R-Diagonal, depending on whether the repeated cell is to the left, above, or diagonally aligned with the blank cell.

Regarding the wrong principle macro category, some specific error types might involve the repetition of a cell that is not adjacent to the blank space (e.g., WP-Copy) or the combination of elements from different cells in a way that does not follow the rules used for creating the matrix.

Some specific errors in the incomplete correlate macro category include changes to the orientation of the correct response (i.e., IC-flip), changes in the color of the correct response (i.e., IC-Neg), changes in the size of the correct response (i.e., IC-Size), or the omission of an element from the correct response (i.e., IC-Inc).

Finally, regarding the difference macro category, specific errors might include a cell being completely white or black (D-Blank) or the merging of different cells within the matrix (D-Union).

The criteria for the classification of the error types were used for the formal definition and

generation of the distractors implemented in the `matRiks` package. These criteria were included in the response options operator with the aim of providing the user with a response list composed of 11 elements (ten distractors and the correct response) among which they could choose the most appropriate ones. Specifically, the response options operator generates a response list composed of the correct response, three repetition distractors, one difference distractor, two wrong principle distractors, and four incomplete correlate distractors. Further details on the formal definition of each of the distractors and on their generation are given in the “Generation of the response list” Section.

### 3 The `matRiks` package

The `matRiks` package (Brancaccio, Epifania, and de Chiusole 2023) can generate  $2 \times 2$  and  $3 \times 3$  Raven-like matrices with their corresponding set of responses (i.e., the correct response and all the distractors described in the Generation of response list section). The Raven-like matrices are generated according to either visuospatial or logic rules, which can be concatenated with three different directional logic, namely vertically, horizontally, and diagonally. Finally, it is possible to print the generated matrices and set of distractors as either single images (i.e., each cell of the matrix and each distractor are printed separately) or as a complete figure with the set of single distractors.

#### Installation

The `matRiks` package is available on CRAN and can be installed as:

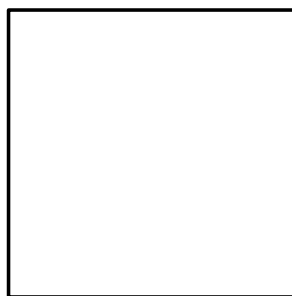
```
install.packages("matRiks")
```

The code `vignette(package = "matRiks")` allows for obtaining the list of all the vignettes included in the package, which illustrates the default figures included in the package and a basic example of its application. Each vignette can be accessed via `vignette("vignette-name", package = "matRiks")`. For instance, the code `vignette("generate_matriks", package = "matRiks")` opens the vignette that contains the instruction on how to generate an RMarkdown file where both the matrix and its associated response options are plotted together.

#### Definition of figures

The `matRiks` package contains several default figures that can be used for the generation of the matrices and of new figures. All figures have class `figure` and they are defined as functions, such that the syntax `figure_name()` must be used to draw them and visualize their structures. The arguments that can be modified inside the parentheses might vary from figure to figure, and they allow for changing different features. The default features of the figures are stored in a list of length 15, and they might vary from figure to figure although the length is unchanged. The features of the figures are defined inside of the list as follows:

- `shape`: character, the name of the figure
- `size.x`: numeric, the length of the semi-major axis of the ellipse within which the figure is inscribed (see the documentation of the **DescTools** for further details)
- `size.y`: numeric, the length of the semi-minor axis of the ellipse within which the figure is inscribed (see the documentation of the **DescTools** for further details)
- `theta.1`: numeric, radians of the rotation of the circle
- `theta.2`: numeric, radians of the rotation of the circle
- `rotation`: numeric, radians of the rotation of the ellipse within which the figure is inscribed
- `pos.x`: numeric, the position on the x-axis
- `pos.y`: numeric, the position on the y-axis
- `lty`: integer, the line type of the margins of the figure
- `lwd`: integer, the width of the margins of the figure
- `num`: numeric, it is equal to 1 if the figure is draw inscribed into an ellipse, and it equal to 2 if the figure is draw as a arc of a circle
- `nv`: integer, the number of vertices of the figures (might vary from 2 – lines – to 100 – circle and ellipse –)
- `shade`: character, the filling of a figure (can also be NA –empty figure–)
- `visible`: integer, the visibility of the figure
- `tag`: character, properties of the figure used for the definition of the distractors



**Figure 3:** A simple square

The tag of each figure describes some generic characteristics of the figure, which are used for the generation of the distractors. For instance, there are tags defining whether the figure is a single figure (`simple`), is composed of two figures (`compose2`) or of 4 figures (`compose4`). Other tags deal with the possibility of changing the shading of the figure (`fill`) or whether the figure can be rotated (`rotate`). The tag of a figure can be seen with code `figure()$tag`. For instance, the tags associated to a square are:

```
square()$tag
```

```
#> [[1]]
#> [1] "simple" "fill"   "d.ext"  "rotate"
```

indicating that the square is a single figure, it can be filled with different shadings, and it can be rotated. The tag `d.ext` specifically refers to the possibility of the figure to be used for the generation of the Difference distractor. Further details are provided in Section XX. The tags associated to malta cross are:

```
#> [[1]]
#> [1] "compose4" "fill"     "d.int"
```

which indicate that the figure is composed of 4 different figures, it can be filled with different shading but it cannot rotate (it can be inferred from the fact that the `rotate` tag is missing).

The `draw` function is designed to print all the figures, as well as the matrices and the response list. To print a figure, the only argument needed inside of the `draw` function is the name of the figure. For instance, a simple square (Figure @ref{fig:square}) can be printed with the command line:

```
draw(square())
```

The figures in `matRiks` are summarized in different categories (e.g., closed figures, black figures, lines), for each of which a vignette lists all the figures. The categories of the figures available in `matRiks`, along with their associated vignettes, are illustrated in Table 1.



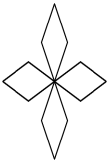




### Concatenation of figures

Other than the pre-existing figures, the `matRiks` package allows for the generation of new figures by concatenating the existing ones. The `cof()` (Concatenation Of Figures) function is designed for this aim. The arguments of the `cof()` function are the names of the default figures presented in the previous section.

For instance, the figure in Figure 4 is obtained by concatenating a `circle()` and a `dot()`.

```
eye <- cof(circle(), dot()) # create the new figure eye by concatenating the circle and the dot
draw(eye)
```

Table 1: List of figures and related vignettes.

Figure Category	Example	Figure Category	Example	Figure Category	Example
Black Figures		Flowers figures		Other figures	
Circle sections		Eight-shaped figures			
Closed figures		Lines			

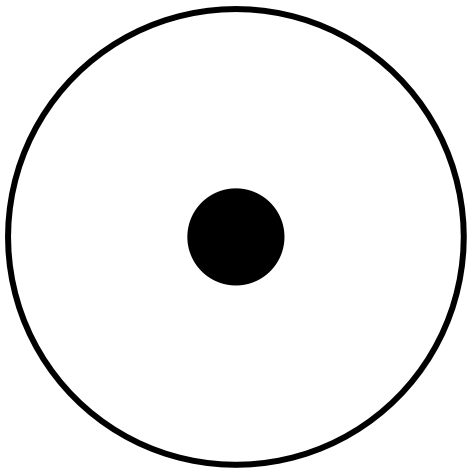


Figure 4: Example of concatenation of circle and dot to obtain an eye-like figure.

**Table 2:** Four-cell matrix

Sq1	Sq3
Sq2	Sq4

**Table 3:** Nine-cell matrix

Sq1	Sq4	Sq7
Sq2	Sq5	Sq8
Sq3	Sq6	Sq9

The resulting figure eye is a named list of lists of class figure. However, the two original figures are still available and can be considered as separate figures, and as such the object eye is considered a concatenation of two figures.

```
eye$shape
```

```
#> [1] "circle" "dot"
```

Function `cof()` has also two optional arguments, namely `single` and `name`, which allow for the generation of new figures. If set to `TRUE`, the first argument forces the outcome of the concatenation to be considered as a new single figure, while the second argument defines a new name for such a figure. The following code recreated the figure in Figure 4 but it forces the new figure to be a single figure named "eye".

```
s_eye <- cof(circle(),dot(),single = TRUE, name = "eye")
s_eye$shape
```

```
#> [1] "eye"
```

Although this difference is trivial from a graphical standpoint (the single figure and the concatenation of two distinct figures appear exactly the same), it is relevant for the application of some of the rules that require a minimum number of figures for their application (e.g., inferential rules).

### Available rules and matrix generation

The function `mat_apply(Sq1, hrules="identity", vrules="identity", mat.type=9)` is the main function for the generation of matrices based on rules. This function allows for the generation of matrices of different dimension, either 4-cell or 9-cell matrices. The dimension of the matrix can be specified with the argument `mat.type`, such that `mat.type = 4` results in 4-cell matrices (Table 2)

and `mat.type = 9` results in 9-cell matrices (default, Table 3).

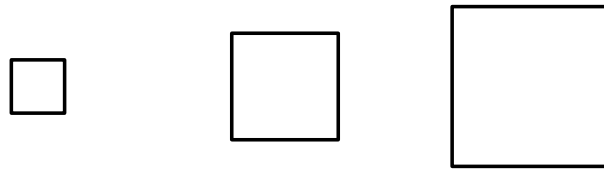
The `Sq1` argument defines the starting figure (i.e., the figure to be plotted in the first cell `Sq1`), which can also be a concatenation of figures. The arguments `hrules` and `vrules` allow for the definition of the directional logic with which the rule(s) is applied, such that the rules specified in `hrules` are manipulated horizontally and those specified in `vrules` are applied vertically.

The application of the `mat_apply()` function results in a named list of class `matriks` that contains the characteristic of the matrix, such as the figures that are visible in each cell, the rule(s) applied horizontally and/or vertically and the type of matrix, either 4-cell or 9-cell. The following example is based on a 9-cell matrix where the identity rule was applied both horizontally and vertically:

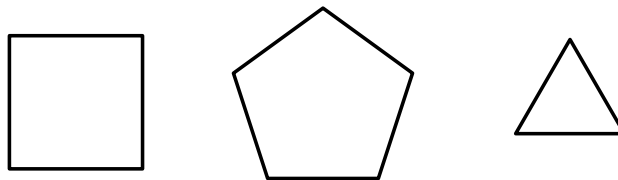
```
#> [1] "Sq1" "Sq2" "Sq3" "Sq4" "Sq5" "Sq6" "Sq7" "Sq8" "Sq9"
#> [10] "hrule" "vrule" "mat.type"
```

In a `matriks` each cell of the matrix is an element of the named list. Specifically, the elements which contain a cell are called `Sq` followed by the number of the cell. In the above example, nine elements (from `Sq1` to `Sq9`) are listed corresponding to the nine cells as depicted in Table 3. In the case of a 4-cells matrix four elements will be present named from `Sq1` to `Sq4`. In addition to the `Sq` elements, the class `matriks` contains `hrule`, `vrule` and `mat.type`. The first two are vectors containing the rule(s) applied horizontally and vertically, respectively. The last one is the dimension of the matrix.

The rules are methods that transform a feature of the figure to obtain a different figure or a figure with a modified feature. The `matRiks` package implements several rules, each of which may



**Figure 5:** Example of incremental rule with reverse application: Change in size



**Figure 6:** Example of permutational rule with reverse application: Change of shape

be classified differently. All the rules functions are characterized by three arguments, namely `fig`, `n`, and `rules`. The `fig` argument specifies the initial figure to which the rule is applied. Some rules, such as `size`, can be applied to any type of figure, while others have specific constraints; for example, the `shape` rule requires concatenation of three figures. Detailed definitions for each rule are provided in Table @ref{tab:rule-types}, under the ‘Definitions’ column.

The `n` argument specifies either the elements of the permutation or the increment value, depending on whether the rule is permutational or incremental. In `mat_apply()`, there is also a link between the cell number and the constant `n`, which ensures that the rule’s directional logic is maintained.

Finally, a single function can apply different rules (see the ‘function’ column in Table XXX). The `rules` argument allows us to specify which specific rule to apply. For example, the `margin` function manipulates line attributes of figures, with options to adjust line width `margin(..., rules = "lwd")` or line type `margin(..., rules = "lty")`. Another use of the `rules` argument is to reverse the default direction of rule application. For instance, `size(..., rules = "size")` typically decreases in size from right to left (see Figure 7). By adding `inv` to the `rules` vector, the rule is applied in reverse order. Figure 5 shows the output of `mat_apply(square(size.x=10), hrules = "size.inv")` applied to a sequence of squares.

The reverse application of the rules can be done with permutational rules as well. By starting with the same set of figures as the one used to generate Figure 8, the specification of the argument `hrules = "shape.inv"` would result in a reverse ordering of the figures across the cells (Figure 6).

Logical rules `AND`, `OR` and `XOR` are a special case rules that cannot be applied in reverse.



**Table 4:** Types of Rules

Rule	Classification	Function	Definition
<b>Identity</b>		identity	Return the original figure without any transformation
<b>AND</b>	logical	logical	Considering the input cell fig, a concatenation of at least three figures is partitioned into three sets: A, B, and C. The rule transforms the figures such that each row or column in the matrix follows a specific sequence of patterns. In these patterns, the first cell displays figures $\{A, C\}$ , the second cell displays $\{A, B\}$ , and the third one exclusively displays figure A. The partitioning of the figures into sets A, B, and C is randomly determined with a random seed based on the n rows/columns of the matrix.
<b>OR</b>	logical	logical	Considering the input cell fig, a concatenation of at least three figures is partitioned into three sets: A, B, and C. The rule transforms the figures such that each row or column in the matrix follows a specific sequence of patterns. In these patterns, the first cell displays figures $\{A, C\}$ , the second cell displays $\{A, B\}$ , and the third one combines all of them in $\{A, B, C\}$ . The partitioning of the figures into sets A, B, and C is randomly determined with a random seed based on the n rows/columns of the matrix.
<b>XOR</b>	logical	logical	Considering the input cell fig, a concatenation of at least three figures is partitioned into three sets: A, B, and C. The rule transforms the figures such that each row or column in the matrix follows a specific sequence pattern. In these patterns, the first elements display figures $\{A, C\}$ , the second display $\{A, B\}$ , and the third display $\{B, C\}$ . The partitioning of the figures into sets A, B, and C is randomly determined with a random seed based on the n rows/columns of the matrix.
<b>line width</b>	permutational	margin	Considering a figure or concatenation of figures, the lwd rule increases the width of the lines in the figure by a constant value n corresponding to the number of rows or columns in the matrix. Therefore, the width can have values 1, 2, or 3 of the default width argument of the R plot. Conversely, the reverse rule lwd.inv decreases the line width by the same quantity.
<b>line type</b>	permutational	margin	Considering a figure or concatenation of figures, the lty rule changes the line type by manipulating the line type argument of the R plot. In the default order with the values of n from 1 to 3, the lines are dashed, dotted, and solid, respectively. Using the reverse rule, with the values of n from 1 to 3, lty.inv has the order dashed, solid, and dotted.

<b>rotate</b>	incremental	rotate	Considering a figure or concatenation of figures and an angle $\theta$ , the rule rotates the figure around its center clockwise by an angle $n\theta$ , where $n$ is the argument $n$ of the function. The value of $\theta$ is equal to $\pi$ divided by any number from 1 to 9 included in the rule argument of the function. For instance, <code>rule=rotation.5</code> sets $\theta = \pi/5$ . By default, $\theta = \pi/4$ . The reverse rule <code>rotation.inv</code> rotates the figure anticlockwise.
<b>size</b>	incremental	size	Considering a figure or concatenation of figures and a constant $k$ , the rule size decreases the figure size proportionally to $nk$ , where $n$ is the argument $n$ of the function. Specifically, the <code>size.x</code> and <code>size.y</code> arguments of the figure are divided by $nk$ . The default value of $k = 0.9$ . The reverse rule <code>size.inv</code> increases the figure arguments <code>size.x</code> and <code>size.y</code> by $nk$ times, with a default value of $k = 0.6$ .
<b>shape</b>	permutational	shape	Considering a concatenation of three single figures denoted A, B, and C, the shape rule permutes which figure is visible in each cell of the matrix. The default order is figures A, B, and C from left to right in the row or from top to bottom in the column. The reverse rule <code>shape.inv</code> has the order C, B, and A.
<b>shade</b>	permutational	shade	Considering a figure or concatenation of figures, the shade rule changes the color of the filling. The argument $n$ of the function goes from 1 to 3 and is mapped into white, grey, and black, respectively. The rule ignores any previous color present in the figure. For instance, when $n=1$ and a figure has <code>shd=black</code> , the application of the rule transforms it into white. There is no reverse rule available at the moment.
<b>multi shade</b>	permutational	shade	Considering a concatenation of figures, the <code>multi.shade</code> rule changes the color of the filling of each figure separately. The rule works exactly as the normal shade, but a random color is assigned to each figure before the transformation. The random color is assigned with the function <code>sample</code> using <code>seed(n)</code> .

The rule classification, originally introduced in the literature as visuospatial and logical, has been modified in the **matRiks** package to focus on constructing matrices rather than solving them. Consequently, the rules are reclassified into incremental and permutational categories.

Figure 7 illustrates the application of the change in size rule on a square. the change in size rule is an incremental rule, in which the size of the square decreases of a fixed quantity in each subsequent object according to a directional logic. Incremental rules apply a fixed increase (or decrease) on each cell of the matrix to obtain the features of the figure in the following cell, starting from the first cell. The order in which the fixed increase (decrease) is applied depends on the directional logic used for the generation of the matrix.

This rule is incremental in the sense that the order of the squares in the cells is determined by the fixed variation in their size, from the larger to the smaller one. If the positions of the squares across the cells is changed, then the definition of the rule is no longer satisfied, unless the squares are ordered from the smaller to the larger. In this case, the rule is still incremental but with a reverse application.

The operation underlying the functioning of the permutational rules is the permutation of the

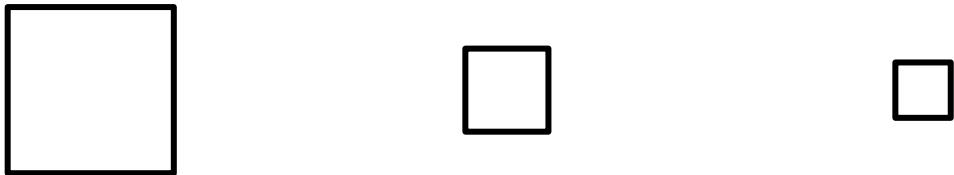


Figure 7: Example of size rule transformation along a row.

figures (or of their features) across the cells according to a directional logic. For instance, consider a matrix where a set of three figures (hexagon(), pentagon(), square()) are manipulated according to the shape rule following an horizontal directional logic (Figure 8).

By default, the starting order of the figures is the order with which the figures are concatenated in the set defined by the user. When the user change the starting order, the rule will adapt consequentially.

It may appear trivial at this point that the transformation obtained from a rule becomes evident only in the comparison between figures. For instance, in Figure 7, the change in size is noticeable only when two cells are considered together. The figure in the first cell is smaller than the one in the second cell, that is smaller than the one in the third and so on. This straightforward concept underlies the creation of a matrix starting from a figure using `mat_apply()`.

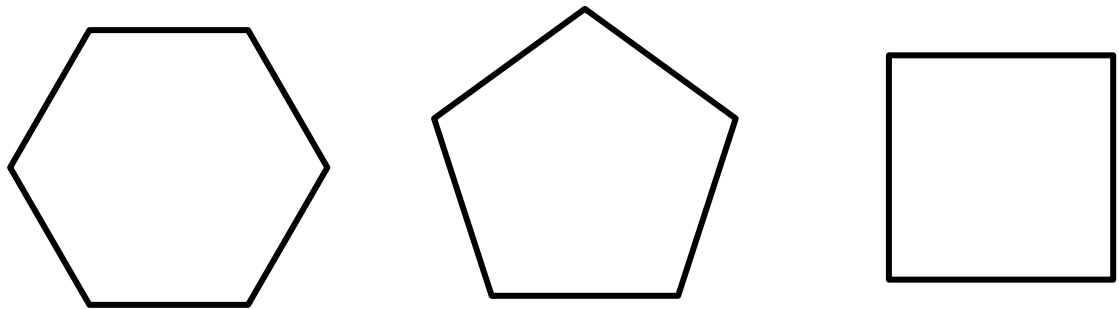
The procedure underlying `mat_apply()` consists of three serial steps. For the sake of simplicity, they will be illustrated for the 9-cell matrices only. The same reasoning applies to 4-cell matrices.

The first step is to generate a named list of all the cells from Sq1 to Sq9. Each cell contains the initial figure defined in Sq1, with no rules applied yet. For example, considering the concatenation of figures `cof(square(), circle(), dot())`, the resulting matrix will be displayed in Figure @ref{fig:examplestep1}.

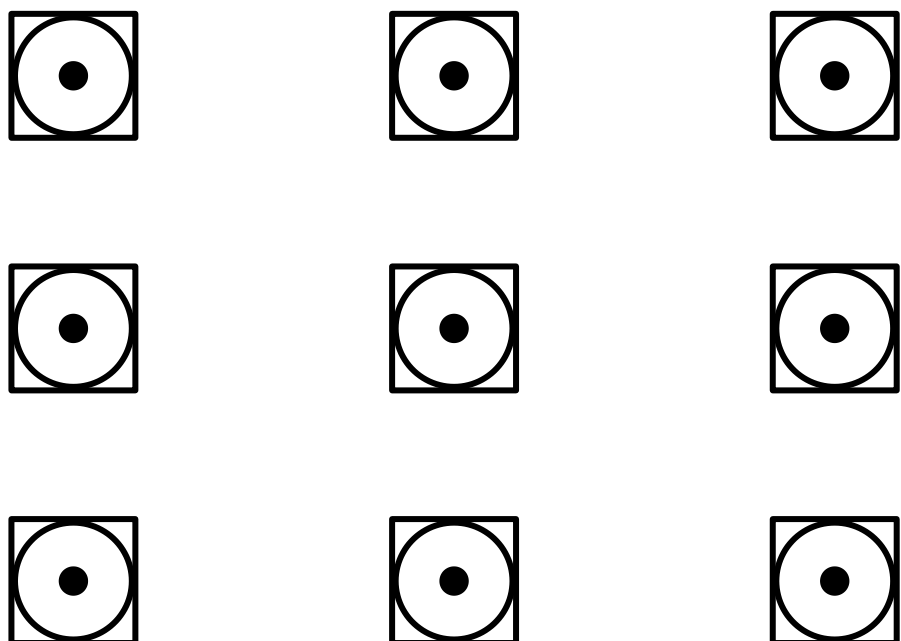
The second step checks whether the arguments `hrules` or `vrules` contain a logical rule. If they do, an error occurs in the following cases: if the logical rule is combined with a visuospatial rule, if a logical rule is used to generate a 4-cell matrix, or if two different logical rules are concurrently applied to the same figures with differing directional logic. Since applying logical rules requires a well-defined relationship between the cells of the matrix, only one logical rule can be applied at a time. If there is a logical rule and none of these error conditions are met, the procedure will generate the entire matrix in a single step.

If there are no logical rules, the procedure enters the third step. The rules in `hrules` are applied following a horizontal directional logic, meaning they are iterated among the elements across columns. Subsequently, the rules specified in `vrules` are applied following a vertical directional logic, meaning they are iterated among the elements across rows.

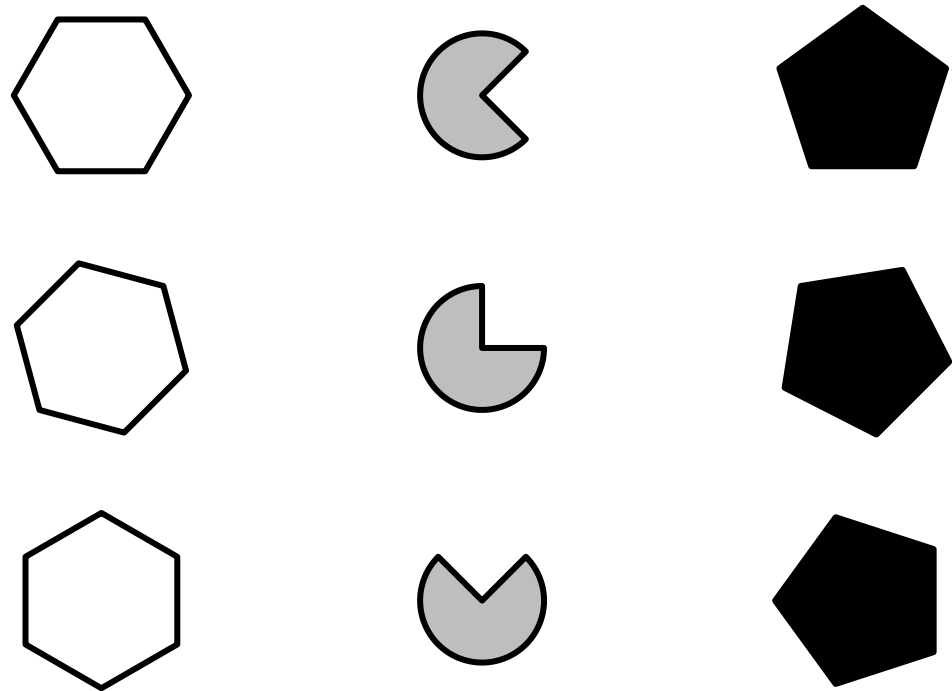
Since the third and forth steps are made one after another using the same rule, for instance `size`, both with a horizontal a vertical logic, the two transformation on the cells ends up to cumulate. For



**Figure 8:** Example of shape rule transformation along a row.



**Figure 9:** Example of outcome of the first step of the `mat_apply` procedure.



**Figure 10:** Single-layer matrix with two rules manipulated horizontally (Shape and filling) and one rules manipulated vertically (Orientation)

instance, considering the code `mat_apply(square(),hrules="size", vrules="size")`, at the end of the third step the figure in Sq5 will be half the size of Sq1. During the forth the size of Sq5 will be halve again resulting in a figure that is a quarter of the initial Sq1.

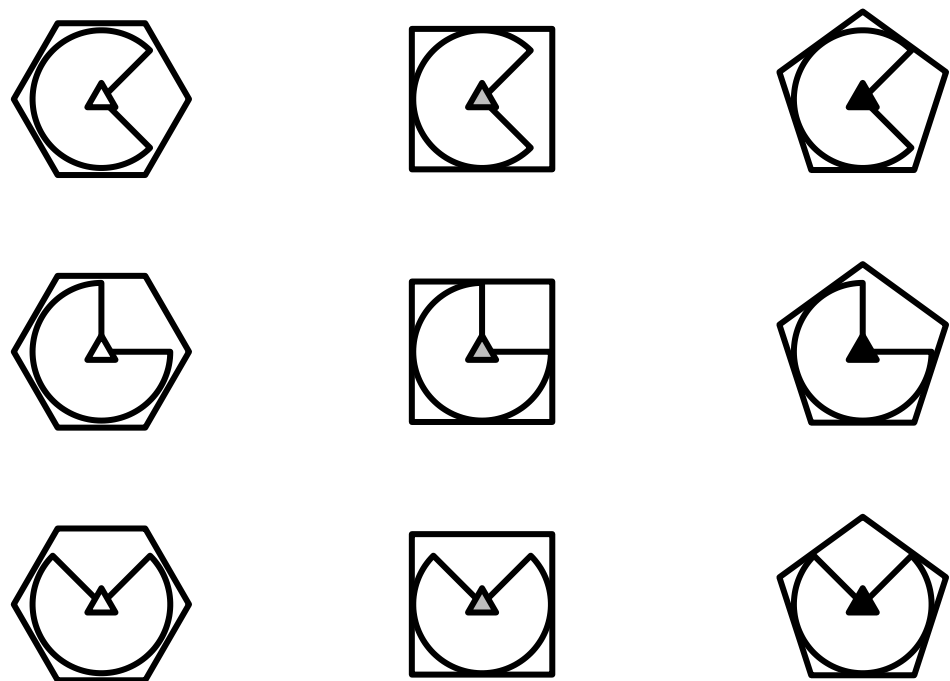
It is worth mentioning that the combination of the same rule horizontally and vertically ends up in the definition of a diagonal directional logic. In particular, if the same rule is applied in reverse in one direction and applied direct in the other, it will result in a TL-LR directional logic. On the other hand if the same rule is direct (or reverse) in both horizontal and vertical directional logic they will give rise to a LL-TR logic.

### Concatenation of matrices

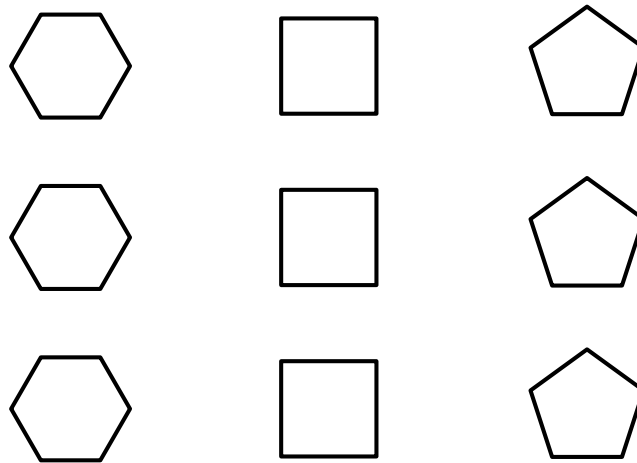
The `com()` (Concatenation Of Matrices) function allows for combining and layering different matrices together to generate a multi-layer matrix.

Figure 10 and Figure 11 depict a matrix created by manipulating the same set of rules. Specifically, two rules are manipulated horizontally (i.e., shade and shape) and one rule is manipulated vertically (i.e., 'rotate'). As such, the figures in the cells and the shading change across columns, while the rotation of the figures change across rows.

The matrix depicted in Figure 10 results from the manipulation of the set of rules to obtain a single-layer matrix, such that all rules are concurrently applied to the same set of figures (i.e., `hexagon()`, `square()`, `pentagon()`). The same set of rules is manipulated to obtain the matrix in Figure 11. However, the three rules are manipulated to obtain three different single-layer matrices, one for each rule (Figures 12, 13, and 14), which are then concatenated together. Each of these single-layer matrices can be considered as a layer of the multi-layer matrix. The layering of the matrices moves from the background (considered as external) to the foreground (considered as internal). Layers are counted inwards  $m = 1, \dots, M$  (where  $M$  is the total number of matrices to concatenate), such that the background matrix is layer 1 and the foreground matrix is layer  $M$ . The matrix in Figure ??(fig:multi-layer) is composed of  $M = 3$  layers: (1) background layer obtained with horizontal manipulation of the shape rule (Figure 12), (2) middle layer obtained with vertical manipulation of the rotate rule



**Figure 11:** Multi-layer matrix with two rules manipulated horizontally (shape and filling) and one rule manipulated vertically (orientation)



**Figure 12:** Layer 1 (Background matrix)

(Figure 13, and (3) foreground layer obtained with horizontal manipulation of the shade rule(Figure 14).

The `com()` function concatenates the single-layer matrices to obtain the multi-layer one:

```
com(multi_a, multi_b, multi_c)
```

The matrices have to be concatenated hierarchically from the background layer to the foreground layer. The hierarchy between the layers is of the uttermost importance for definition of the distractors, specifically for the generation of the incomplete correlate ones. Generally, the manipulation of the correct response to obtain the incomplete correlate distractors is applied to the figures in the foreground layer *M*. Further details on the definition and on the generation of the incomplete correlate distractors of multi-layer matrices are given in section Response list.

### Generation of the response list

The generation of the distractors composing the response list is constrained to: (i) the type of matrix (i.e., single-layer matrix vs. multi-layer matrix), (ii) the rule(s) manipulated for the matrix generation, and (iii) the directional logic of the rule manipulation. To the best of our knowledge, there is not a formal definition of the specific features of each distractor and on their applicability given the above-mentioned constrained.

As such, the development of the response option operator required the formal definition of the distractors and of all the possible exceptions given the constraints imposed by the matrices generated via the matrix operator. The definitions of the distractors implemented in the **matRiks** package is reported in Table 5.

The response option operator (implemented in the `response_list()` function) runs through different steps. First, it checks for the number of rules and the directional logic with which that have been manipulated, as well as whether the matrix is a single matrix or a multi-layer matrix. This check is crucial for the generation of the incomplete correlate distractors, but not for the other types of distractors (i.e., R, WP, and D) since they are generated by considering the figures in a cell as a single figure. Conversely, since the IC distractors are a modification of a single feature of the correct response, there is the need to determine whether they can be generated in the first place and, if so, on which element of the correct response the change should be applied.

In single-layer matrices, the IC-inc distractor cannot be generated given that it requires the removal of an element of the figure, while all other IC distractors can be generated, unless a figure with specific tags is employed. For instance, the s-shaped figures such as the `miley()` cannot change color, and, as such, the "shade" tag is not present in their tag list. When the response option generator does not find the shade tag among the tags of `miley()` to generate the IC-neg distractor, it throws a warning and the IC-neg distractor is replaced by the correct response canceled out by a black thick cross.

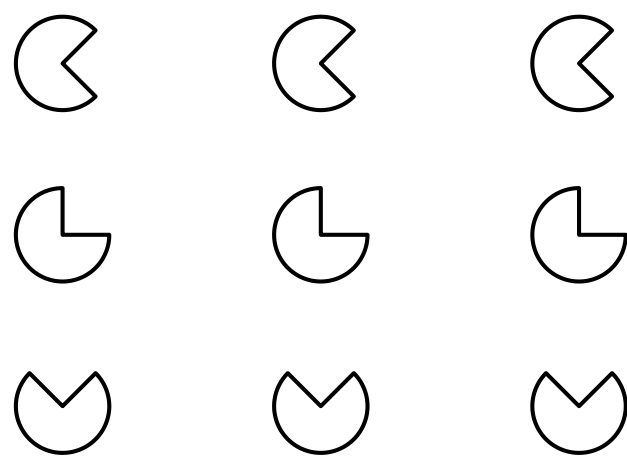


Figure 13: Layer 2

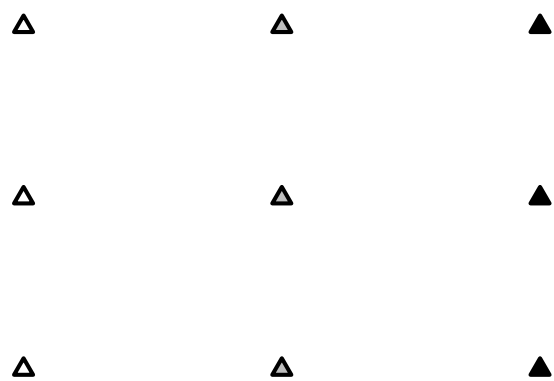


Figure 14: Layer 3 (Foreground matrix)



**Table 5:** Definition of the distractors implemented in the **matRiks** package for  $3 \times 3$  and  $2 \times 2$  matrices

Distractors	$3 \times 3$ matrices	$2 \times 2$ matrices
R-Left	SQ8	SQ3
R-Top	SQ6	SQ2
R-diag	SQ5	SQ1
Wp-Copy	SQ1 or SQ3	SQ1
WP-Matrix	SQ1 or SQ3 with the superimposition of another cell.	SQ3 or SQ2 with the superimposition of the rotation of WP-Copy
Difference	SQ1 or SQ3, SQ4, SQ7 with the superimposition of a figure which is not manipulated in the matrix.	SQ3 or SQ1 with the superimposition of a figure that is not manipulated in the matrix
IC-Inc	It is the correct response with a missing element	Same as $3 \times 3$ matrices
	Single-Layer: Not possible	Same as $3 \times 3$ matrices
	Multi-layer: The most internal figure is removed from the correct response.	Same as $3 \times 3$ matrices
	Logic matrices: The element that is removed is randomly selected.	Same as $3 \times 3$ matrices
IC-Neg	Color inversion of the correct response (single-layer matrix) or of one of its figures (multi-layer matrix)	Same as $3 \times 3$ matrices
	Single-layer matrix: Color inversion of the figure in the correct response	Same as $3 \times 3$ matrices
	Multi-layer matrix: Color inversion of the most internal figure of the correct response	Same as $3 \times 3$ matrices
IC-Flip	Rotation or reflection of the correct response (single-layer matrix) or of one of its figures (multi-layer matrix)	Same as $3 \times 3$ matrices
	Single-layer matrix: Reflection/Rotation of the figure in the correct response	Same as $3 \times 3$ matrices
	Multi-layer matrix: Reflection/Rotation of the most internal figure of the correct response	Same as $3 \times 3$ matrices
IC-Scale	Resize of the correct response (single-layer matrix) or of one of its figures (multi-layer matrix)	Same as $3 \times 3$ matrices
	Single-layer: Resize of the figure in the correct response	Same as $3 \times 3$ matrices
	Multi-layer matrix: Only the most internal figure in the correct response is resized	Same as $3 \times 3$ matrices

In multi-layer matrices, the IC distractors are generated by modifying the figure in the foreground matrix. Before applying the manipulation, the tags of the figure in the foreground matrix are checked to investigate whether all the IC distractors can be generated from that figure. If this tests false, then the response option generator moves towards the background of the multi-layer matrix, until it finds a figure whose tags satisfy the condition for creating all the IC distractors.

For the generation of WP and D distractors, the choice between using cell SQ1 or cell SQ3 strictly depends on the number of rules and on the directional logic with which they are manipulated. Specifically, cell SQ1 is selected when the matrix is generated via the manipulation of a single rule with V, H, or TL-LR directional logic. If the manipulation of the single rule follows a LL-TR directional logic or two rules are manipulated with all possible directional logic, the SQ3 cell is chosen. This is done to avoid that the WP-Copy distractors can be interpreted as IC distractors. In some instances, the R distractors in  $3 \times 3$  matrices cannot be generated because they are equal to the correct response. For instance, if a matrix is generated with the vertical manipulation of a single rule, the R-Left distractor (SQ8) is equal to the correct response. In such instances, the distractor that is equal to the correct response is generated by the response options operator but it is covered with a thick black cross.

A similar procedure is applied for choosing which cell to use between SQ3 and SQ2 or between SQ3 and SQ1 for the generation of WP-Matrix and Difference distractors in  $2 \times 2$  matrices. In both cases, the choice depends on whether at least one rule is applied both horizontally and vertically. If the same rule is applied both horizontally and vertically, then SQ3 is chosen, otherwise SQ2 and SQ1 are chosen.

The response options operator generates the response list (i.e., the correct response and all the distractor presented in the Table) through the `response_list()` function, which results in a list of figures that correspond to each of the distractors. For instance, the distractors of the matrix in Figure 10 can be obtained with:

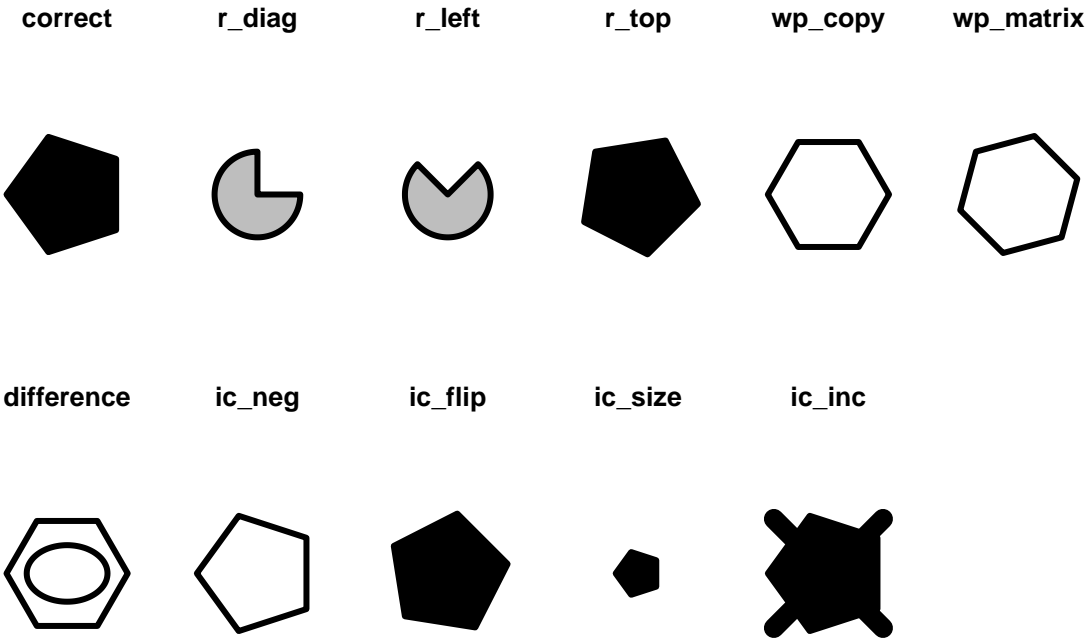
```
response_list(single_matrix)
```

The function `response_list()` results in a named list of length 11 containing all the response options:

```
#> [1] "correct" "r_diag" "r_left" "r_top" "wp_copy" "wp_matrix" "difference" "ic_neg"
#> [9] "ic_flip" "ic_size" "ic_inc"
```

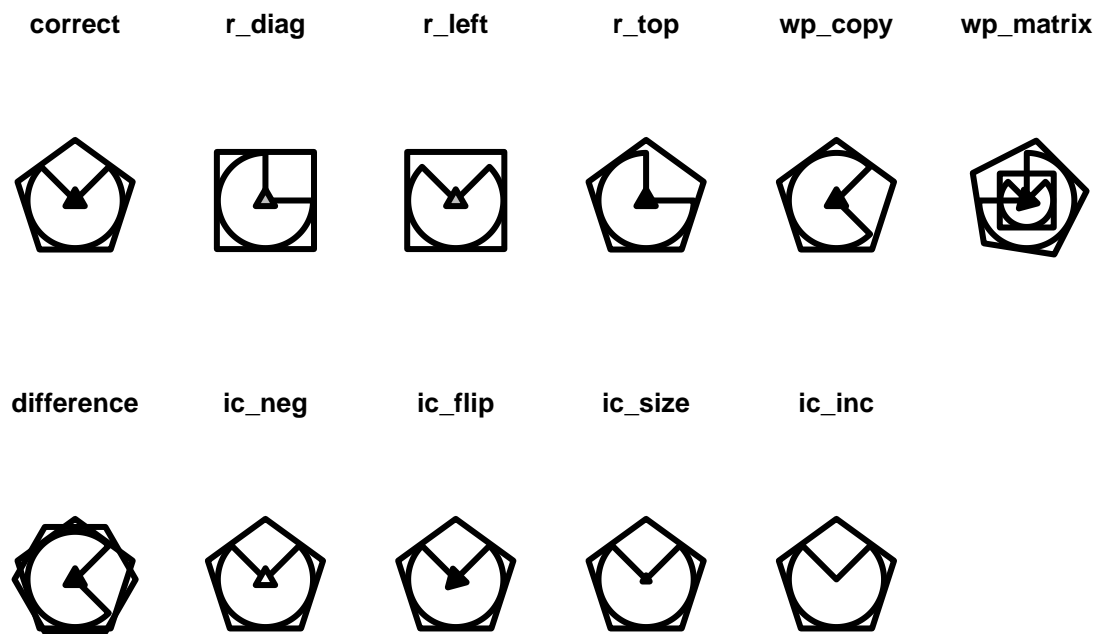
and it can be plotted as:

```
#> Warning in ic_inc.matriks(obj): IC-Inc cannot be obtained with a single figure
```



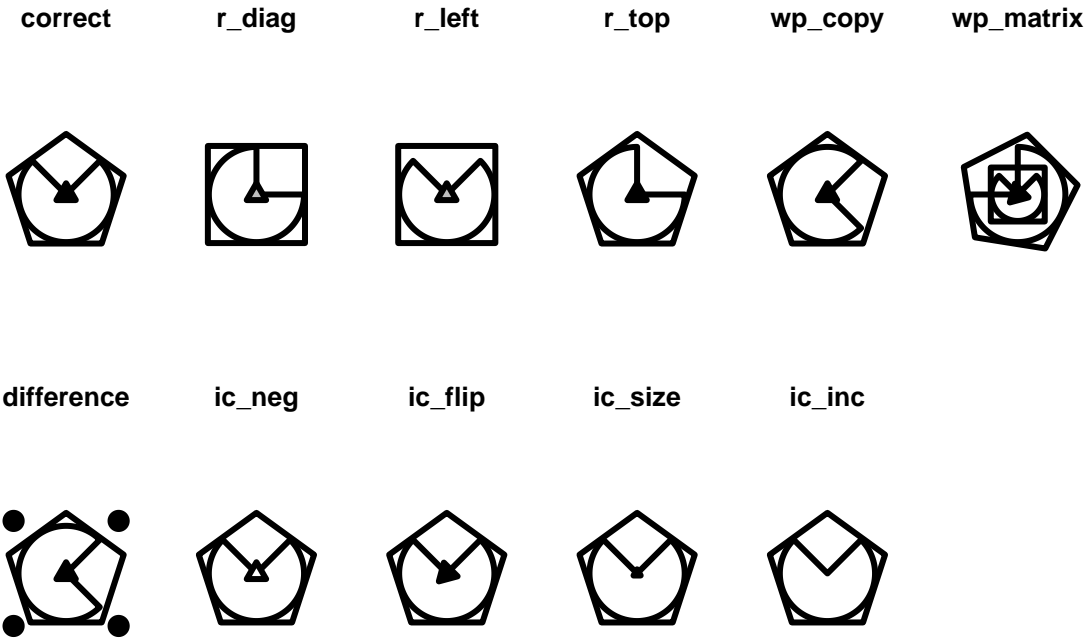
As it can be noted, a warning has appeared and it refers to the IC-Inc distractor. As per Table 5., the distractor IC-Inc is defined as the correct response with a missing element. However, having a single-layer matrix does not allow for the removal of any element, hence the warning “IC-Inc cannot be obtained with a single figure” is thrown and the IC-Inc distractor is replaced by the correct response over which a black thick cross is imposed.

Given that the matrix in Figure 11 is composed of three layers, it is possible to obtain also the IC-Inc distractor:



However, the difference distractor is not well defined. The user can change the random seed for the generation of this distractor with the argument `seed()`, such that another random figure is chosen among the available ones:

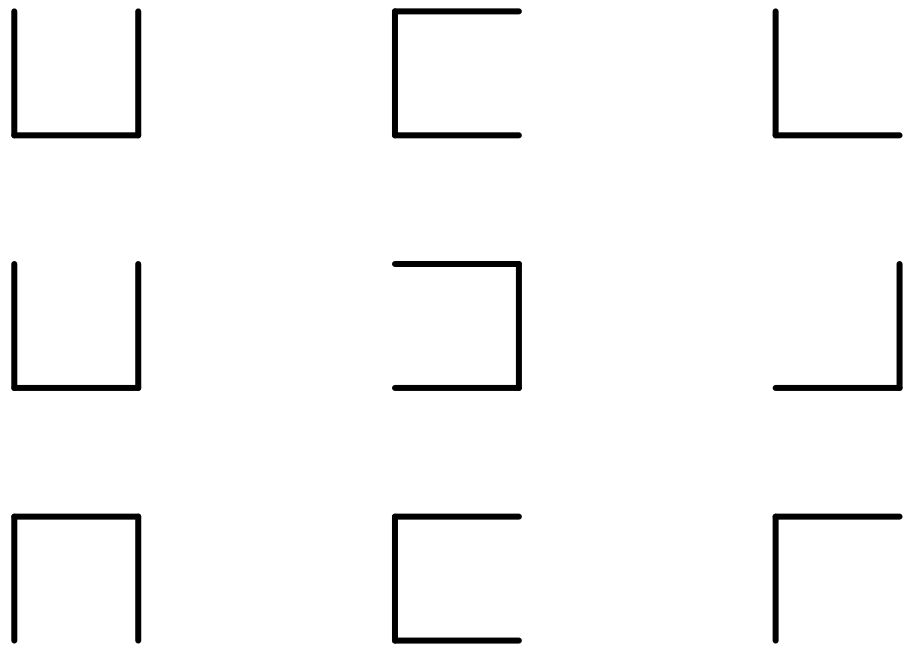
```
draw(response_list(multi_matrix, seed = 7),
      main = TRUE)
```



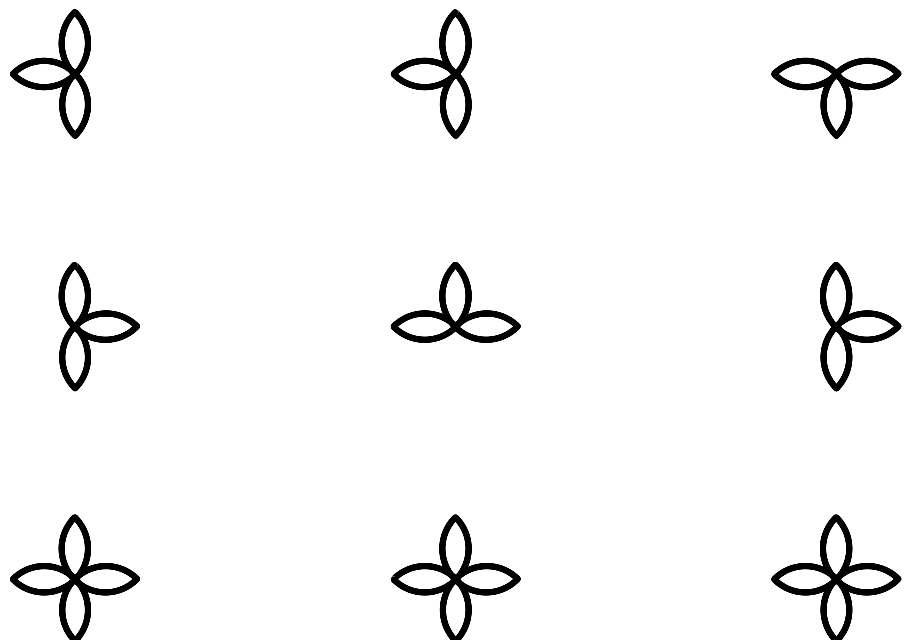
A complete example

This section presents a complete example on how to generate a multi-layer  $3 \times 3$  matrix with logical rules and its related response options.

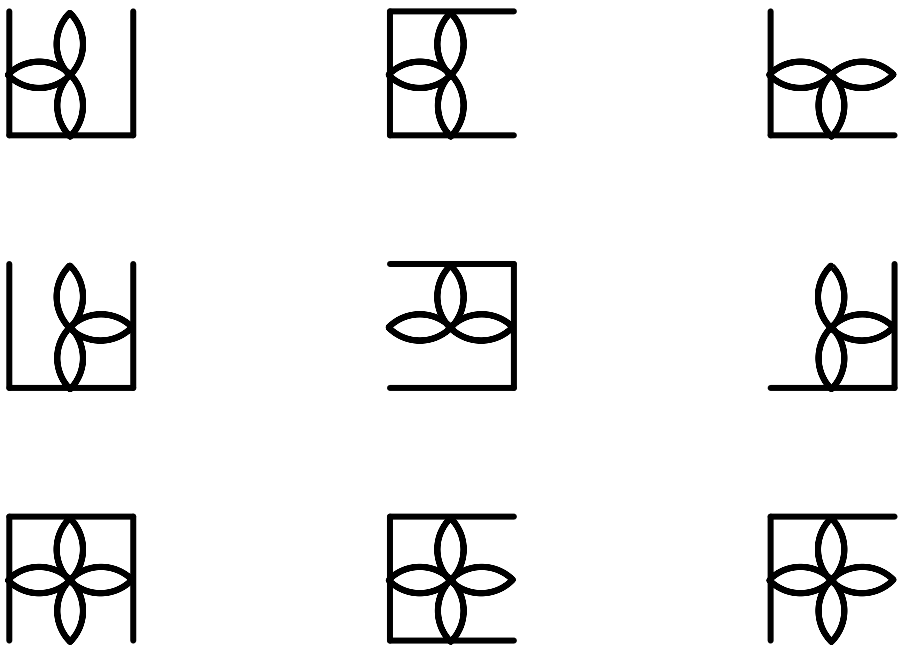
The first layer is generated by manipulating the AND logical rule according to an H directional logic on a square composed of 4 lines:



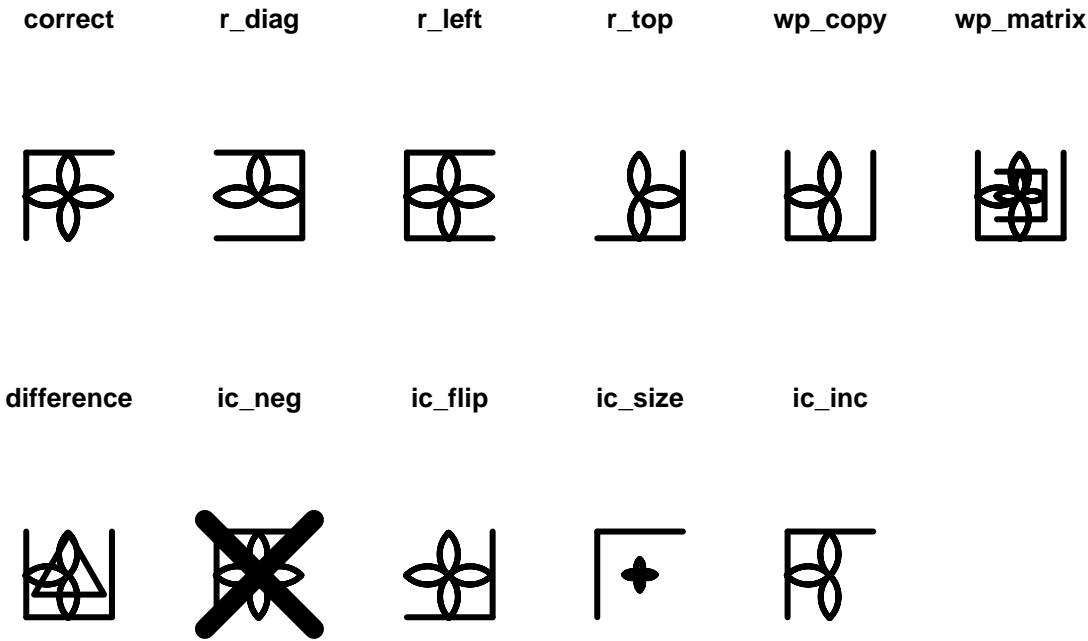
The second layer is generated by manipulating the OR logical rule according to a V directional logic on a flower composed of 4 petals:



The multi-layer matrix can be composed with the `com()` function by concatenating the two single-layer matrices:

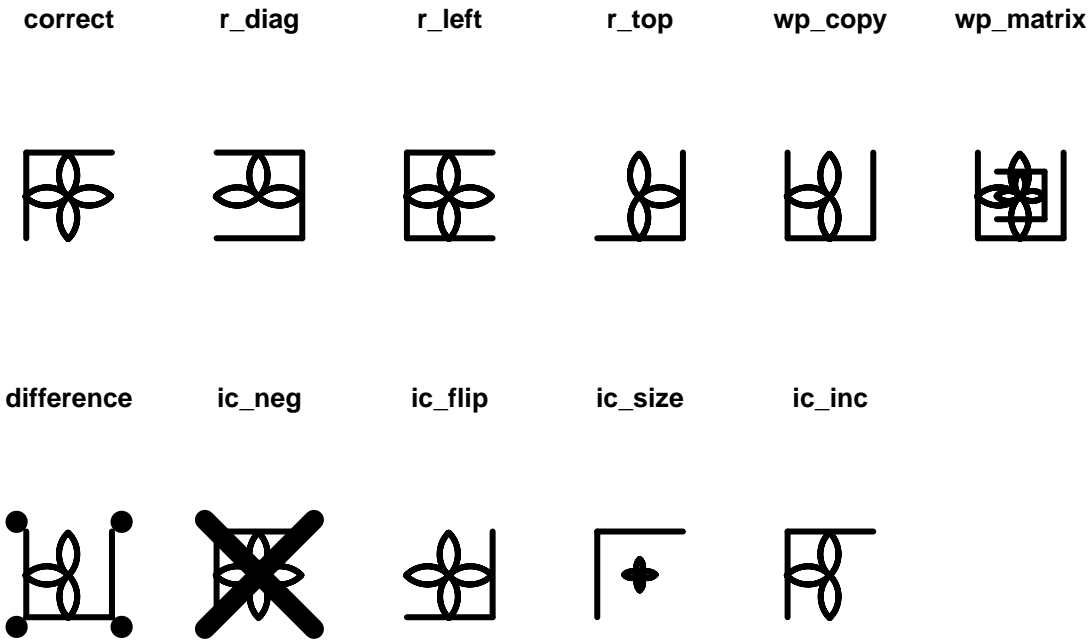


The response options associated with the multi-layer matrix can be generated with the `response_list()` function:



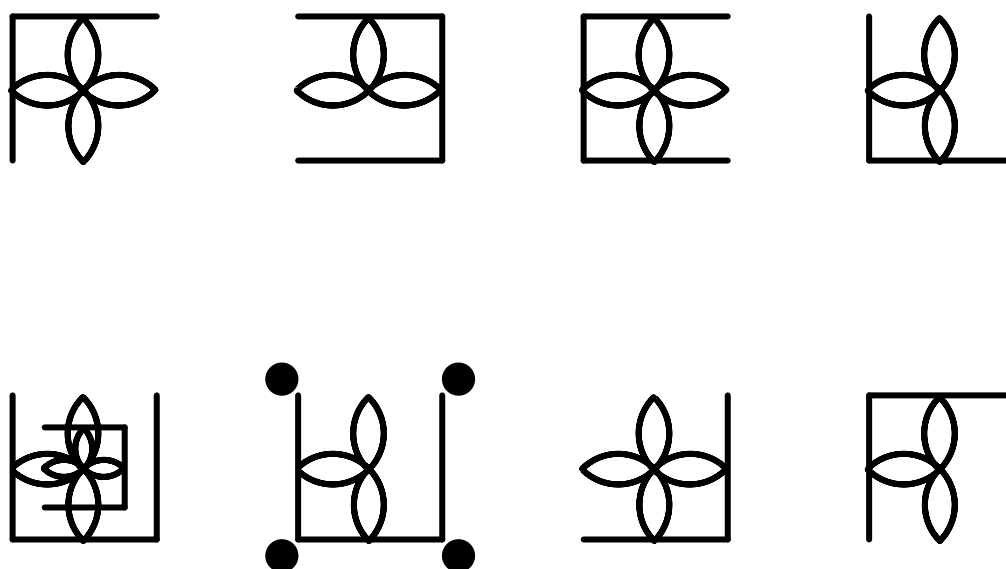
Since the filling color the miley cannot be changed, the IC-Neg distractor cannot be generated and the function throws a warning and the `ic_neg` distractor is replaced by the correct response with

the superimposition of a thick black cross. The difference distractor does not look good. The figure that is superimposed to the cell taken from the matrix can be changed by changing the seed in the `response_list()` function:



Assuming that a response list of length 8 (the correct response along with seven distractors) is associated with the multi-layer matrix, a character vector with the labels of the chosen distractors can be specified directly in the `draw` function:





## 4 Summary

This article briefly illustrates the functioning of the **matRiks** package for the automatic generation of Raven-like stimuli. This package has been developed with the intention of providing the users with a flexible, open-source, and easy-to-use tool for generating Raven-like matrices according to different rules, encompassing both visuo-spatial and logical rules, along with their associated response list.

The rules for generating the matrices and for generating the distractors associated to each matrix that are implemented in the **matRiks** package derive from vast literature concerning Raven's matrices and the error patterns observed on the CPM and the APM. As such, this package potentially provide the possibility for generating stimuli that are equivalent in terms of rules and response options to the standard Raven's stimuli. This allows for a comparison between the responses observed with the standard stimuli and those generated with the package.

This package has been developed within a broader project founded by the Italian Ministry of University and Research, which is aimed at the development of an intelligent system for the adaptive assessment of executive functions and fluid intelligence among general and clinical populations. The test used for assessment of fluid intelligence (named MatriKS) was developed following the principles of Raven's Matrices. Specifically, the stimuli composing the test have been developed with the **matRiks** package (mdpi).

This package allows for high degrees of freedom in the generation of matrices with different difficulty. For instance, it allows for the generation of stimuli that combine multiple rules by exploiting the possibility of layering multiple matrices together. Moreover, the high degree of control provided to the users allow them to thoroughly manipulate the difficulty of the matrix, for instance by adding or removing objects and combining together different rules.

The package is regularly maintained and new functions will be available in the future. As such, the users should refer to the official documentation of the package that is constantly updated.

Although the functions implemented in **matRiks** are quite straightforward and easy to use, they need a basic knowledge of the R language. To overcome this issue and allow for a wider use of the package even among people that are not familiar with R, a web application built with the **shiny** package (shiny) will be developed in the future.

## References

- Brancaccio, Andrea, Ottavia M. Epifania, and Debora de Chiusole. 2023. *matRiks: Generates Raven-Like Matrices According to Rules*. <https://CRAN.R-project.org/package=matRiks>.
- Kunda, Maithilee, Isabelle Soulières, Agata Rozga, and Ashok K. Goel. 2016. "Error Patterns on the Raven's Standard Progressive Matrices Test." *Intelligence* 59 (November): 181–98. <https://doi.org/10.1016/j.intell.2016.09.004>.
- Raven, John, and H. Raven. 2004. "Manual for Raven's Progressive Matrices and Vocabulary Scales, Section 3: The Standard Progressive Matrices, Including the Parallel and Plus Versions. 2000 Edition, Updated 2004." In, Section 3.
- Sievert, Carson. 2020. *Interactive Web-Based Data Visualization with r, Plotly, and Shiny*. Chapman; Hall/CRC. <https://plotly-r.com>.

Ottavia M. Epifania  
 Univerisity of Trento  
 Department of Psyhcolology and Social Science  
 Rovereto, Italy  
<https://www.britannica.com/animal/quokka>  
 ORCID: 0000-1721-1511-1101  
[qqquo@ulm.edu](mailto:qqquo@ulm.edu)

Bounciest Bilby  
 University of Little MatesUniversity of Aussie Animals  
 Department of Letter Q, Somewhere, Australia  
 Department of Marsupials, Somewhere, Australia  
<https://www.britannica.com/animal/bilby>  
 ORCID: 0000-0002-0912-0225  
[bbil@ulm.edu](mailto:bbil@ulm.edu)